Introduction to the EASE 2016 Special Section: Evidence-Based Software Engineering: Past, Present, and Future

Sarah Beecham[a], David Bowes[b], Klaas-Jan Stol[a]

[a]*Lero—The Irish Software Research Centre*
University of Limerick, Ireland
[b]*University of Hertfordshire*
United Kingdom

Abstract

The International Conference on Evaluation and Assessment in Software Engineering (EASE) had its twentieth anniversary in 2016, with that year's edition hosted in Limerick, Ireland. Founded in 1997, the EASE conference was the first event solely dedicated to encouraging empirical research in software engineering, and its founders have been longtime advocates of evidence-based software engineering (EBSE). In this editorial, we briefly look back at the history of EBSE and the EASE conference. We then introduce the four articles which are revised and extended versions of papers presented at EASE 2016. We conclude by looking at the future of EBSE, and provide some suggestions for conducting and reporting empirical research.

Keywords: Evidence-based software engineering, Empirical research

1. Introduction

This Special Section marks the twentieth edition of the International Conference on Evaluation and Assessment in Software Engineering (EASE), which was held on 1-3 June 2016 in Limerick, Ireland. In recognition of this milestone, we include a short history of how EASE began and evolved into the premier conference for empirical researchers we see today. We also take this opportunity to reflect on the evolution of Empirical Software Engineering as a research field in its own right, and whether the vision of a group of academics who kick-started this conference series has been realised.

This Special Section includes extended versions of four of the best technical track research papers that were presented at the 20th edition of the EASE conference. The remainder of this editorial is laid out as follows. In Section 2 we present a brief history of the evidence-based movement in software engineering. Section 3 presents the articles that were selected for inclusion in the special section. In Section 4, we present our outlook on the future of evidence-based software engineering.

2. The Past: A Brief History of Evaluation and Assessment in Software Engineering

EASE began life back in 1997 when four researchers from Keely University in the UK, Barbara Kitchenham, Pearl Brereton, David Budgen and Steve Linkman, recognised a need for a conference concentrating on evaluation aspects of software engineering. In the words of Barbara Kitchenham [1]:

> At the time, the International Conference on Software Engineering (ICSE) was not so interested in Empirical Software Engineering, and the International Software Metrics Symposium had limited scope.

Keele University hosted the EASE conference for the following six years. During that time, EASE grew from a small, locally based conference, operating a single track, with a workshop-like atmosphere, to an international conference that attracted researchers from all over the globe.

Those early pioneers had a clear vision. They wanted the conference to be inclusive, affordable and to attract young, novice researchers by providing an informal workshop atmosphere. Costs were kept down by the group doing all the organising and administration themselves. Although early versions of EASE proceedings were not published, the quality of paper submissions was acknowledged, and the best papers from EASE were published in special journal issues [2, 3]. This raised the profile of the conference, which became recognised by leading software engineering academics, as in 2004, thanks to a suggestion put forward by Anthony Finkelstein, EASE moved from Keele University to Edinburgh, where it was co-located with ICSE. Since then, EASE has been hosted by various software engineering research groups around Europe, South America, and China.

Since the early beginnings of the EASE conference, it has seen considerable growth in participation. The 2016 conference held in Limerick, Ireland, was a truly global affair with delegates attending from 32 countries. Papers were presented across four tracks to include: technical papers, industry papers, tool papers, and short and work-in progress papers. The main technical track reported the latest empirical research as well as systematic reviews, and kept firmly to the original vision with a focus on evidence-based software engineering. Papers include original work on testing, fault prediction, software project management,

metrics, requirements engineering process, software engineering process, and industry experience.

The year 2004 also saw the publication of a seminal paper by EASE conference co-founder Kitchenham et al. [4], entitled "Evidence-Based Software Engineering" in the International Conference on Software Engineering. This paper set out a vision of evidence-based software engineering (EBSE), inspired by evidence-based medicine. Interestingly, this vision, initially embodied in the EASE conference, now attracted significant attention at ICSE—arguably, the point where evidence-based software engineering became mainstream within the wider software engineering research community.

Dybå et al. [5] have argued that EBSE provides a method for fruitful cooperation between research and practice. Realizing the EBSE vision requires a close link between research and practice so that research is relevant to practitioners' needs, and practitioners are willing to participate in research. Conducting industry relevant research lies at the heart of much of the empirical research we undertake today [6].

3. The Present: Articles in this Special Section

Kitchenham et al. [4] suggest the goal of EBSE should be:

> to provide the means by which current best evidence from research can be integrated with practical experience and human values in the decision making process regarding the development and maintenance of software.

We complement this overall goal by specifying four concrete objectives of EBSE to serve the software industry:

- **To developing deep understanding**: To employ a variety of methods so as to enrich our understanding of techniques, practices, and phenomena in the software engineering area. Such studies are needed to understand the state of practice, and to analyze and document complex phenomena (see, for example, Smite et al. [7]);
- **Lead to insights that are Practical and Meaningful**: To identify outcome measures that are practical and meaningful to practitioners [4];
- **Support Assessment and Evaluation**: To conduct studies that generate evidence to help decide whether techniques, practices and processes actually work in practice. Studies of the use of pair programming are good examples that achieve this objective (see for example Di Bella et al. [8]);
- **Support decision making**: To conduct studies that can help software organisations to make better decisions. Important decisions such as hiring more developers are often made at an executive level; project managers may need to show evidence for their need for more staff (see, for example, Fitzgerald et al. [9]).

Each of the four articles included in this Special Section has achieved these objectives in specific ways, which we summarize in Table 1.

The first paper, "The Effects of Perceived Value and Stakeholder Satisfaction on Software Project Impact," by Hennie Huijgens, Arie van Deursen, and Rini van Solingen investigates stakeholder satisfaction in software projects. The authors observe that traditional notions of project success and failure are limited, and that an alternative way to evaluate projects is more fruitful. Huijgens and colleagues specifically focus on capturing customer value and link this to traditional project metrics such as cost, duration, defects, and size. Using a mixed methods approach, the authors draw on quantitative and qualitative data from 26 industry projects in two organisations. Huijgens et al. conclude that *" 'within time and cost' does not automatically lead to satisfied stakeholders."* We believe this paper is important because it balances careful quantitative analysis with qualitative insights on a topic that is highly relevant to the software industry: providing value to customers.

The second paper, "Risk-averse Slope-based Thresholds: Definition and Empirical Evaluation," by Sandro Morasca and Luigi Lavazza, is an example of statistically determining threshold values of metrics for given scenarios. In this paper, the authors demonstrate that their approach works well for determining thresholds for code metrics used to predict defects. Furthermore, they show that the technique works and can be used even when the software under development is still in an immature state. The paper is a must-read for any researcher who has a set of metrics and needs to have a robust methodology for justifying a particular threshold.

The third paper, "Findings from a Multi-method Study on Test-driven Development," by Davide Fucci, Simone Romano, Giuseppe Scanniello, Burak Turhan, Markku Oivo, and Natalia Juristo, lifts the lid on what actually takes place when we say developers are carrying out test-driven development. The study shows that developers of different levels of software development experience approach code review, testing, and development in typically non-textbook approaches. Their findings highlight the need for code refactoring and general maintenance of not only production code, but also of code which tests the system.

Finally, the fourth paper, "Benefits and Limitations of Project-to-Project Job Rotation in Software Organizations: A Synthesis of Evidence," by Ronnie Santos, Fabio Silva, and Cleyton de Magalhães, addresses an important issue of software project management. This empirical study investigates the managerial practice of job rotation as a means of reducing job monotony, boredom, and exhaustion that can result from job simplification, specialization, and repetition. Taking a mixed methods approach that involved a synthesis of evidence taken from their literature review and two industrial case studies, the authors found a good number of benefits to job rotation, but also many limitations to this approach, with some factors falling into both categories depending on their context. They conclude that job rotation as a managerial practice can yield important job outcomes, such as motivation and satisfaction, and can decrease job monotony and job burnout. However, there are also limitations to changing jobs when moving projects that project managers should be

Table 1: Objectives of Evidence-Based Software Engineering and Articles in this Special Section

Article	Develop understanding	Practical and Meaningful	Assessment and Evaluation	Decision Making
Huijgens et al.: The Effects of Perceived Value and Stakeholder Satisfaction on Software Project Impact	Investigates concept of stakeholder satisfaction and perceived value, using both quantitative and qualitative data.	Study uses concepts and metrics that are meaningful to case study companies.	Evaluation of relationship between stakeholder satisfaction, perceived value w.r.t. traditional measures of project success.	Findings can help managers to identify course of action to improve stakeholder satisfaction.
Morasca & Lavazza: Risk-averse Slope-based Thresholds: Definition and Empirical Evaluation	Develop understanding of how thresholds can be set while complying with a number of desirable properties.	Practitioners can use the proposed method for setting so-called "early symptom" thresholds.	Empirical validation of practicality of a risk-averse approach to setting thresholds using the PROMISE data set.	Managers may use the findings to identify potentially faulty modules, at different levels of risk-aversion.
Fucci et al.: Findings from a Multi-method Study on Test-driven Development	Reveals how developers carry out Test-Driven Development in practice.	Different levels of developer experience used to develop theory on Test-Driven Development.	Drawn directly from practice, using multi method approach to strengthen findings.	Consider the need to refactor code and general maintenance of code to test the system.
Santos et al.: Benefits and Limitations of Project-to-Project Job Rotation in Software Organizations: A Synthesis of Evidence	A combination of evidence from literature and a qualitative study helps bring a new understanding to the concept of job rotation.	The problem of job monotony is well motivated, and through careful analysis, this study provides a solution to this issue.	Context is provided and depth of qualitative analysis (combined with evidence from the literature) compensates for the small sample.	Providing both pros and cons of implementing this practice can help project managers decide whether or not to apply this technique.

aware of, such as the likelihood that it can lead to an increase in intra-group social conflicts, individual cognitive effort, and can temporarily decrease productivity.

4. The Future: Some Recommendations for Reporting EBSE Studies

On the basis of our experience as reviewers and editors, as well as as insights from other authors [10, 11], we offer a set of recommendations to consider when engaging in evidence-based software engineering research. Most readers will be well aware of these suggestions, but, we believe, having checklist by way of a reminder is helpful nevertheless.

4.1. Paper Structure

One of the most important aspects of any paper is its structure. In our experience as reviewers and editors, a poor structure is often a reason for a major revision—or worse. The "classic" structure defines the following sections: introduction, background and related work, research method, results, discussion, and conclusion. Deviating from this classic structure often leads to illogical ordering of material, for example:

- An introduction section that contains too much information about related work, methodology, and results. While summarising information should be provided, an introduction should not contain too many details;
- A related work section that contains a 'pre-study' that helps motivate the study (any study performed by an author should be strictly separated from a discussion of others' work);
- A results section that contains too much speculation, which should be left for a discussion section. We acknowledge that the presentation of results may be combined with discussion, but the key point is to not confuse the "what is" (the findings) from "what might be" (speculation and potential implications);
- A related work section at the end of the paper, where its main purpose has passed—a point we return to below.

While we acknowledge that each paper is unique, we strongly suggest authors start with the classic structure, and only change it if necessary.

4.2. Literature Review

A background section should discuss research that is relevant to the current study. A common shortcoming—in our view—is that the background section is "tacked on" without clearly building on the related work. Merely listing all papers that have conducted studies on a given topic is not an adequate literature review. Even if it covers a comprehensive list. Bem captured this well [12, p. 172] (as cited by Webster and Watson [13]) by noting that:

> authors of literature reviews are at risk of producing mind-numbing lists of citations and findings that resemble a phone book – impressive case, lots of numbers, but not much plot.

Furthermore, to publish a literature review as a standalone piece of research requires taking insights and findings from the body of knowledge and synthesizing it in a way that produces new knowledge. A detailed systematic review might also indicate that certain questions remain unanswered, leading to the need to conduct additional studies to fill certain "gaps in knowledge," or might challenge current assumptions [14, 15], or might lead researchers to replicate a study. In any case, a review as a standalone must produce new knowledge through synthesis.

We also observe that many authors add a related work section at the end of a paper (to be ignored by many readers), which is, to use a Dutch proverb, akin to offering *"mustard after the meal."* A related work section should describe the state of the art, or the current state of knowledge so as to position and motivate a study. (Because motivation is such an important aspect of a paper, we discuss this in more detail in Sec. 4.3 below.) At the end of a paper, there is no need to motivate the study. Of course, linking the new results of a paper back to extant work, so as to contrast, compare, and establish a "delta" of the new study fits well in a discussion section, after presentation of the results.

4.3. Motivation

Conducting research is a resource-intensive activity, whether the research is done empirically or through literature studies. Each year, thousands of papers are published—yet, the impact on practice of most papers seems to be minimal [16]. Before undertaking any research study, it is therefore important to consider the motivation for a study which represents a considerable investment of resources. The following is a checklist to consider before embarking on new studies:

- *What is the problem exactly, and why is it important?* What problem will be solved? Clearly, "problems" can be either practical or conceptual—not all research needs to address industry challenges; conducting thorough conceptual research resulting in new theoretical lenses or conceptual frameworks to look at the world are important, too, as they might challenge long-held assumptions.
- *Why is the proposed research interesting?* Note that this is a very subjective point—what is highly interesting to some may be less interesting to others. We suggest a useful way to think about this is to ask oneself the question: given a "Top 5" of interesting questions, is *this* the one I should be investigating?
- *What work has been done so far to address the problem?* Has the problem been solved already? What are shortcomings of extant work? To what extent can the newly proposed research build on extant work? In other words, *is more research really necessary?* Note that just because little or no research has been done in a given area, does not in itself mean it should be done.
- *Finally, are our assumptions about the problem correct?* Have we captured the essence of the problem, and are we as researchers doing the right thing? For example, when Open Source Software (OSS) became commercially viable, researchers *assumed* that practitioners needed thorough evaluation frameworks so that those OSS components could be fairly evaluated. Unfortunately, software developers do not operate in such a rational manner—the assumption that they did, and consequently the assumptions underpinning the proposed solutions, were wrong. In this context, we'd like to echo Karl Wiegers [17]: *"Read My Lips: No New Models!"*

4.4. Substantiate your claims

In a keynote given by Barbara Kitchenham in the early days of EASE, an area of concern was the temptation for researchers to make "sound-bite generalisations" [11]. When making claims about their results, researchers need to be careful that any bold statements are supported by evidence from their empirical studies or from the literature (when motivating their studies).

Some basic heuristics to help ensure that the message is clear and supported, as adapted from Levy et al. [18], include:

- *Structure* – as noted above. Use a reliable structure that is explicit and follows proper argumentation.
- *Define terms* used in the study using clear examples, and back them up with high quality peer-reviewed sources.
- *Keep focused* by providing a reason for including the information given as support. If using the literature to motivate the study, do you return to these findings in your discussion? (cf. Sec. 4.3)
- *Substantiate your assumptions*: make assumptions explicit. Use only assumptions that are free of subjective judgment and are based on valid reasoning. These can often take the form of a hypothesis.
- *Avoid Fallacies* such as generalization, abstraction and misplaced concreteness. Bob Glass, emeritus editor-in-chief of the *Journal of Systems and Software*, highlighted the many fallacy traps that we in software engineering research fall into [19], such as claiming certain tools and technologies bring orders of magnitude improvements, when at best the improvement might be between 5-35% [19, p.19].
- *Check Quality of Evidence* where possible, *"use reliable documented evidence from quality peer-review sources that is legitimate and relevant, not trivial"* [18].

4.5. Replication

Replication could play a key role in Empirical Software Engineering, by *"allowing the community to build knowledge about which results or observations hold under which conditions"* [20]. Yet, all too often it is not possible for others to replicate studies, due to incomplete documentation of the methods, context, and raw data used in the study. A recurring omission is the use of a survey to collect data, where the authors fail to include the survey questions. In order to allow replication there needs to be adequate documentation; this documentation should include sufficient details about the setting in which the study was conducted to allow replication. Where space doesn't allow for inclusion of this supporting material, a useful approach is to provide such additional information in a technical report that is permanently archived and accessible, and referenced in the main paper.

4.6. Industry Relevance

EBSE is an applied research field, and as a branch of software engineering research, strives towards *"the transfer and widespread use of research results in industry"* [21]. This goes to the very heart of EBSE, in which researchers apply techniques to support industry who *"are often under pressure to adopt immature technologies because of market and management pressures"* [5]. However, conducting such research does not guarantee that the recommendations, or validation of technologies, will reach its intended audience [6]. We as a community need to be more proactive and work closely with industry from the outset [16]. Engaged scholarship and employing methods such as Action Research are strategies that encourage and foster strong relationships with industry partners to solve practical problems in rigorous ways. One excellent example of this is a recent study of testing practices in the automotive sector [22].

This call for relevance goes beyond empirical studies, and also relates to Systematic Literature Reviews, that also need to have industry relevance [23]. As a reminder, *"The aim of an SLR is not just to aggregate all existing evidence on a research question; it is also intended to support the development of evidence-based guidelines for practitioners"* [24]. While the synthesis phase of an SLR is perhaps the most difficult part (which would explain why it is often poorly executed), some authors in the software engineering domain have offered useful advice [25, 26, 10].

4.7. Evaluating Quality of Data

It is a simple thing, but one we rarely do: check the validity of the data being used. As mentioned in the section on replication, do we have sufficient information in order to be able to trust the data? Can we actually check the validity of the data by inspection. Hall and Bowes [27] show that data quality is a key issue which is not discussed in papers describing the use of machine learning for defect prediction. Liebchen and Shepperd [28] reveal that we are still not checking our data eight years after their first major description of data quality issues(!). Ghotra et al. [29] show that using cleaned data has a significant effect on the performance of machine learning techniques in the area of software defect prediction.

Just because the data are available and used by others, are they really valid? Always ask: *"How did the researchers who provide the data obtain them in the first place?"* The following are some suggestions that can act as a starting point for checking data quality issues in papers, though we readily admit this list is by no means complete:

- Counts should be unsigned integers [30] (i.e., greater than or equal to zero);
- Lines of code should be less than the length of the file [31] (though cumulative code changes or code churn may exceed the length of the file);
- When two different measurement techniques have been applied, the number of measurements from each set should be the same;
- When joining two data sets together with a one to one mapping, the cardinality of all sets should be the same
- Check that the results published by other authors have the same number of classes, methods, and files as you do.

In machine learning, folklore asks *"Will the real Iris dataset stand up please?"* [32] suggesting that there are multiple versions of the Iris data set. In defect prediction, we can now ask *"will the real NASA defect datasets please stand up!"*, because different people have derived their own versions (with and without errors).

4.8. Concluding Thoughts

We humbly offer these suggestions with an open admission that we, too, are guilty of many, if not all of the above critiques. Conducting and reporting research well are skills that require much time to perfect—but as any author who has worked tirelessly on a paper knows, finishing a research study is highly rewarding. We hope that our suggestions will help to improve the state of practice of evidence-based software engineering. Returning to the question we posed in the introduction section: after twenty years, has the vision of the group of pioneering empirical researchers who kick-started EASE been realised? We conclude by paraphrasing Barbara Kitchenham [1]:

> Evidence-Based Software Engineering remains a goal but one we seldom achieve. Systematic reviews support EBSE, but we should never make the mistake of believing they are synonymous.

Acknowledgments

We thank *Information and Software Technology's* Special Content Editor, Prof. Claes Wohlin for his continuous support and patience. We also thank the anonymous reviewers of the four included papers, all of which benefited from your detailed feedback. We also thank Dr. John Noll for proofreading an earlier draft of this editorial. Last but not least, we thank Prof. Barbara Kitchenham for her guidance throughout the process, from selection of the included papers, right through to writing this editorial. This work was supported, in part, by Science Foundation Ireland grants 13/RC/2094 and 15/SIRG/3293 and co-funded under the European Regional Development Fund through the Southern & Eastern Regional Operational Programme to Lero—the Irish Software Research Centre (www.lero.ie).

References

[1] Barbara Kitchenham, personal communication, April 2017.
[2] B. Kitchenham, P. Brereton, D. Budgen, S. Linkman, V. Almstrum, S. Pfleeger, Evaluation and assessment in software engineering, Inform. Softw. Technol. 39 (1997) 731–734.
[3] S. Linkman, Evaluation and assessment in software engineering 1998, Inform. Softw. Technol. 40 (1998) 793–794.
[4] B. A. Kitchenham, T. Dybå, M. Jørgensen, Evidence-based software engineering, in: Proc. 26th International Conference on Software Engineering (ICSE), IEEE Computer Society, 2004, pp. 273–281.

[5] T. Dybå, B. Kitchenham, M. Jørgensen, Evidence-based software engineering for practitioners, IEEE Softw. 22 (1) (2005) 58–65.
[6] S. Beecham, P. O'Leary, S. Baker, I. Richardson, J. Noll, Making software engineering research relevant, Computer 47 (4) (2014) 80–83.
[7] D. Šmite, N. B. Moe, A. Šāblis, C. Wohlin, Software teams and their knowledge networks in large-scale software development, Inform. Softw. Technol. 86 (2017) 71–86.
[8] E. Di Bella, I. Fronza, N. Phaphoom, A. Sillitti, G. Succi, J. Vlasenko, Pair programming and software defects–a large, industrial case study, IEEE Trans. Softw. Engineer. 39 (7) (2013) 930–953.
[9] B. Fitzgerald, M. Musiał, K. Stol, Evidence-based decision making in lean software project management, in: Proc. International Conference on Software Engineering (SEIP Track), ACM, 2014, pp. 93–102.
[10] C. Wohlin, Writing for synthesis of evidence in empirical software engineering, in: Proc. 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), 2014.
[11] B. Kitchenham, Five things I hate about empirical software engineering. keynote address. ease conference. keele university, 2002.
[12] D. Bem, Writing a review article for *Psychological Bulletin*, Psychological Bulletin 118 (2) (1995) 172–177.
[13] J. Webster, R. T. Watson, Analyzing the past to prepare for the future: Writing a literature review, MIS Quarterly 26 (2) (2002) xiii–xxiii.
[14] M. Alvesson, J. Sandberg, Generating research questions through problematization, Academy of Management Review 36 (2) (2011) 247–271.
[15] K. Rolland, B. Fitzgerald, T. Dingsøyr, K. Stol, Problematizing agile in the large: Alternative assumptions for large-scale agile development, in: Proc. 37th International Conference on Information Systems (ICIS), 2016.
[16] S. Beecham, P. O'Leary, I. Richardson, S. Baker, J. Noll, Who are we doing global software engineering research for?, in: Proc. IEEE 8th International Conference on Global Software Engineering, 2013, pp. 41–50.
[17] K. E. Wiegers, Read my lips: No new models!, IEEE Softw. 15 (5) (1998) 10–13.
[18] Y. Levy, T. J. Ellis, A systems approach to conduct an effective literature review in support of information systems research, Informing Science: International Journal of an Emerging Transdiscipline 9 (1) (2006) 181–212.
[19] R. L. Glass, Facts and fallacies of software engineering, Addison-Wesley Professional, 2002.
[20] F. J. Shull, J. C. Carver, S. Vegas, N. Juristo, The role of replications in empirical software engineering, Empir. Softw. Eng. 13 (2) (2008) 211–218.
[21] M. Ivarsson, T. Gorschek, A method for evaluating rigor and industrial relevance of technology evaluations, Empir. Software Eng 16 (3) (2011) 365–395.
[22] A. Kasoju, K. Petersen, M. Mäntylä, Analysing an automative testing process with evidence-based software engineering, Inform. Softw. Technol. 55 (7) (2013) 1237–1259.
[23] S. Anderson, P. Sagar, B. Smith, K. Wallace, What we have learnt adopting evidence-based software engineering for industrial practice, in: Proc. 20th International Conference on Evaluation and Assessment in Software Engineering (EASE), ACM, 2016.
[24] B. Kitchenham, O. P. Brereton, D. Budgen, M. Turner, J. Bailey, S. Linkman, Systematic literature reviews in software engineering–a systematic literature review, Inform. Softw. Technol. 51 (1).
[25] D. Cruzes, T. Dybå, Research synthesis in software engineering: A tertiary study, Inform. Softw. Technol. 53 (2011) 440–455.
[26] D. Cruzes, T. Dybå, P. Runeson, Höst, Case studies synthesis: a thematic, cross-case, and narrative synthesis worked example, Empir. Softw. Engineer. 20 (6) (2015) 1634–1665.
[27] T. Hall, D. Bowes, The state of machine learning methodology in software fault prediction, in: Proc. 11th International Conference on Machine Learning and Applications, Vol. 2, 2012, pp. 308–313.
[28] G. Liebchen, M. Shepperd, Data sets and data quality in software engineering: Eight years on, in: Proc. 12th International Conference on Predictive Models and Data Analytics in Software Engineering, ACM, 2016.
[29] B. Ghotra, S. McIntosh, A. E. Hassan, Revisiting the impact of classification techniques on the performance of defect prediction models, in: Proc. 37th International Conf. on Software Engineering (ICSE), 2015.
[30] G. D. Boetticher, Improving credibility of machine learner models in software engineering, Advanced Machine Learner Applications in Software Engineering (Series on Software Engineering and Knowledge Engineering) (2006) 52–72.
[31] J. Petrić, D. Bowes, T. Hall, B. Christianson, N. Baddoo, The jinx on the NASA software defect data sets, in: Proc. 20th International Conference on Evaluation and Assessment in Software Engineering, EASE '16, ACM, New York, NY, USA, 2016.
[32] J. C. Bezdek, J. M. Keller, R. Krishnapuram, L. I. Kuncheva, N. R. Pal, Will the real iris data please stand up?, IEEE Transactions on Fuzzy Systems 7 (3) (1999) 368–369.